DynexSolve: The Dynex Platform Proof-of-Useful-Work Scheme

Dynex Developers

February 1, 2023
V.1.0

Abstract

Dynex is a next-generation platform for neuromorphic computing based on a groundbreaking flexible blockchain protocol. It consists of participating Proof-of-Useful-Work (PoUW) miners that constitute a decentralised neuromorphic supercomputing network which is capable of performing computations at unprecedented speed and efficiency – even exceeding quantum computing.

The Dynex neuromorphic chips run on miners' computers and are used to solve computational tasks, that's the algorithm called DynexSolve. Every progress miners contribute is stored in the blockchain.

All this is wrapped in a sustainable and long term business model. Dynex customers are corporates, organisations or research bodies that are not able to solve their complex computational problems due to lack of computing power and high energy consumption of traditional computer networks. These DynexSolve projects are priced with a certain amount of Dynex coins. DynexSolve algorithm is the first mining algorithm which solves real-world computational problems while providing Proof-of-Useful-Work during the mining process.

This document describes the DynexSolve algorithm used on the Dynex Platform since December 1, 2022.

Rationale

Herein, there will be many references to modern computers and the computation they perform. It is important to realise that computing is fundamentally a **physical process**[1]. The statement may seem obvious when considering the physical processes harnessed by the electronic components of computers (for example, transistors), however, virtually any physical process can be harnessed for some form of computation. Note, that we are speaking of **Alan Turing's model of computation**[2], that is, a mapping (transition function) between two sets of finite symbols (input and output) in discrete time.

[1] Massimiliano Di Ventra and Fabio L. Traversa. Perspective: Memcomputing: Leveraging memory and physics to compute efficiently. Journal of Applied Physics, 123(18):180901, 2018.

[2] Michael R. Garey and David S. Johnson. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

It is important to distinguish between continuous and discrete time: **Dynex chips** operate in **continuous time**[3], though, their **simulations** on modern computers **require the discretisation** of time. Continuous time is physical time: a fundamental physical quantity. Discrete time is not a physical quantity, and might be best understood as counting time: counting something (function calls, integration steps, etc.) to give an indication (perhaps approximation) of the physical time. In the literature of Physics and other Physical Sciences, physical time has an assigned SI unit of seconds, whereas in Computer Science and related disciplines, counting time is dimensionless.

Granting the infinite resources utilised by a Turing machine[4], universal Dynex chips (UDCs) have been shown to be Turing-complete, meaning universal Dynex chips can simulate universal Turing machine. The universal Dynex chip class contains digital and analog machines. While analog Dynex chips theoretically have tremendous computational power, analog systems cannot be engineered for scalability, as their growing size requires growing resources to achieve the same accuracy. It is the **digital Dynex chip** that is **scalable**, and the focus of this algorithm.

The Fourth Missing Circuit Element

Modern computers rely on the implementation of uni-directional logic gates that represent Boolean functions[5]. Circuits built to simulate Boolean functions are desirable because they are deterministic: A unique input has a unique, reproducible output.

Modern computers relegate the task of logic to central processing units (CPUs). However, the resources required for the task might exhaust the resources present within the CPU, specifically, cache memory. For typical processes on modern computers, random-access memory (RAM) is the memory used for data and machine code, and is external to the CPU. The physical separation of CPU and RAM results in what is known as the **von Neumann bottleneck**, a slow down in computation caused by the transfer of information between physical locations[6].

To overcome the von Neumann bottleneck, we propose computing with and in memory, utilising **ideal memristors**[7]. Distinct from in-memory computation[8], it is an efficient computing paradigm that uses memory to process and store information in the same physical location.

[3] A physical process is necessarily continuous in time, as discrete time is not physical, rather a necessary consequence of simulating a physical system.

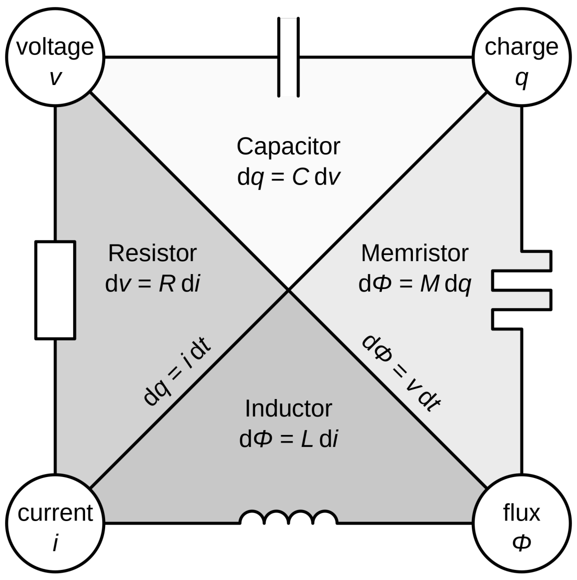
[4] A physical realization of Dynex chips will, of course, have finite resources. However, the “chips” used to study computational complexity are theoretical and impossible to build

[5] Behrooz Parhami. Computer arithmetic, volume 20. Oxford university press, 2010.

[6] John Backus. Can programming be liberated from the von neumann style?: A functional style and its algebra of programs. Commun. ACM, 21(8):613–641, August 1978.

[7] Chua, L. (1971). "Memristor-The missing circuit element". *IEEE Transactions on Circuit Theory*. 18 (5): 507–519.

[8] H. Zhang, G. Chen, B. C. Ooi, K. Tan, and M. Zhang. In-memory big data management and processing: A survey. IEEE Transactions on Knowledge and Data Engineering, 27(7):1920–1948, 2015.



Conceptual symmetries of resistor, capacitor, inductor, and memristor

A **memristor** is an electrical component that limits or regulates the flow of electrical current in a circuit and remembers the amount of charge that has previously flowed through it. Memristors are important because they are non-volatile, meaning that they **retain memory without power**.

The original concept for memristors, as conceived in 1971 by Professor Leon Chua at the University of California, Berkeley[9], was a nonlinear, passive two-terminal electrical component that linked electric charge and magnetic flux (**“The missing circuit element“**). Since then, the definition of memristor has been broadened to include any form of non-volatile memory that is based on resistance switching, which increases the flow of current in one direction and decreases the flow of current in the opposite direction.

A memristor is often compared to an imaginary pipe that carries water. When the water flows in one direction, the pipe's diameter expands and allows the water to flow faster -- but when the water flows in the opposite direction, the pipe's diameter contracts and slows the water's flow down. If the water is shut off, the pipe retains its diameter until the water is turned back on. To continue the analogy, when a memristor's power is shut off, the memristor retains its resistance value. ***This would mean that if power to a computer was cut off with a hard shut down, all the applications and documents that were open before the shut down would still be right there the screen when the computer was restarted****.*

[9] L. Chua, "Memristor-The missing circuit element," in *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507-519, September 1971, doi: 10.1109/TCT.1971.1083337.

Memristors, which are considered to be a sub-category of resistive RAM, are one of several storage technologies that have been predicted to replace flash memory. Scientists at HP Labs built the first working memristor in 2008 and since that time, researchers in many large IT companies have explored how memristors can be used to create smaller, faster, low-power computers that do not require data to be transferred between volatile and non-volatile memory.

A **digital Dynex chip** is realised as a **memristor based bi-directional logic circuit**. These circuits differ from traditional logic circuits in that input and output terminals are no longer distinct. In a traditional logic circuit, some input is given and the output is the result of computation performed on the input, via uni-directional logic gates. In contrast, a memristor based bi-directional logic circuit can be operated by **assigning the output** terminals, then **reading the input** terminals.

Operating a logic circuit "backwards" has many applications. An example is integer factorisation: Given an integer, factor it into its prime factors. For simplicity, assume the given integer, b, is the product of two prime numbers, p and q. If given p and q, then a multiplication circuit can be employed to find the product, b, of the two prime numbers. A traditional logic circuit, appropriately designed, can easily perform this task. Now, if given b and told it can be factored into two prime numbers, we take the same multiplication circuit structure (logic gates connected similarly), but design it to be a memristor based bi-directional logic circuit so the logic gates become terminal agnostic, meaning signal can be received and sent on any terminal of the logic gate. However, the new logic gates are not bijective, so the entire circuit will have to self-organiae to produce the values of p and q on the "input" terminals.

Self-organising logic is a recently-suggested framework that allows the solution of Boolean truth tables "in reverse," i.e., it is able to satisfy the logical proposition of gates regardless to which terminal(s) the truth value is assigned ("terminal-agnostic logic"). It can be realised if time non-locality (memory) is present. A practical realisation of self-organising logic gates can be done by combining circuit elements with and without memory. By employing one such realisation, it can be shown numerically, that self-organising logic gates **exploit elementary instantons** to reach equilibrium points. Instantons are classical trajectories of the non-linear equations of motion describing self-organising logic gates, and connect topologically distinct critical points in the phase space. By linear analysis at those points it can be shown that these instantons connect the initial critical point of the dynamics, with at least one unstable direction, directly to the final fixed point. It can also be shown that the memory content of these gates only affects the relaxation time to reach the logically consistent solution. By solving the corresponding stochastic differential equations, since instantons connect critical points, noise and perturbations may change the instanton trajectory in the phase space, but not the initial and final critical points. Therefore, **even for extremely large noise levels**, the gates **self-organise to the correct solution**.

Note that the self-organising logic we consider here has no relation to the invertible universal Toffoli gate that is employed, e.g., in quantum computation[10]. Toffoli gates are truly one-to-one invertible, having 3-bit inputs and 3-bit outputs. On the other hand, self-organising logic gates need only to satisfy the correct logical proposition, without a one-to-one relation between any number of input and output terminals. Instead, it is worth mentioning another type of bi-directional logic that has been recently discussed in[11] using stochastic units (called p-bits). These units fluctuate among all possible consistent inputs. However, in contrast to that work, the invertible logic we consider here is **deterministic**.

With time being a fundamental ingredient, a **dynamical systems approach** is most natural to describe such gates. In particular, non-linear electronic (non-quantum) circuit elements with and without memory have been suggested as building blocks to realise self-organising logic gates in practice[12].

By assembling self-organising logic gates with the appropriate architecture, one then obtains circuits that can **solve complex problems efficiently** by mapping the equilibrium (fixed) points of such circuits to the solution of the problem at hand, as shown in [13,14,15,16]. Moreover, it has been proved that, if those systems are engineered to be point dissipative[17], then, if equilibrium points are present, they do not show chaotic behaviour[18] or periodic orbits[19].

It was subsequently demonstrated[20], using topological field theory (TFT) applied to dynamical systems, that these circuits are described by a Witten-type TFT[21], and they

[10] Tommaso Toffoli. Reversible computing. In International Colloquium on Automata, Languages, and Programming, pages 632–644. Springer, 1980.

[11] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta. Stochastic p-bits for invertible logic. Phys. Rev. X, 7:031014, Jul 2017.

[12] Fabio Lorenzo Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27:023107, 2017.

[13] Fabio Lorenzo Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27:023107, 2017.

[14] H. Manukian, F. L. Traversa, and M. Di Ventra. Memcomputing numerical inversion with self-organizing logic gates. IEEE Transactions on Neural Networks and Learning Systems, PP(99):1–6, 2017.

[15] Haik Manukian, Fabio L Traversa, and Massimiliano Di Ventra. Accelerating deep learning with memcomputing. Neural Networks, 110:1–7, 2019.

[16] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. Ann. Phys. (Berlin), 529:1700123, 2017.

[17] J.K. Hale. Asymptotic Behavior of Dissipative Systems, volume 25 of Mathematical Surveys and Monographs. American Mathematical Society, Providence, Rhode Island, 2nd edition, 2010.

[18] M. Di Ventra and F. L. Traversa. Absence of chaos in Digital Memcomputing Machines with solutions. Physics Letter A, 2017.

[19] M. Di Ventra and F. L. Traversa. Absence of periodic orbits in digital memcomputing machines with solutions. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27:101101, 2017.

[20] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. Ann. Phys. (Berlin), 529:1700123, 2017.

[21] E. Witten. Topological quantum field theory. Comms. in Math. Phys., 117:353–386, 1988.

support long-range order, mediated by instantons. Instantons are classical trajectories of the non-linear equations of motion describing these circuits (see, e.g., [22] or [23]).

The intrinsic non-locality of instantons, coupled with the topological character of critical points, is reminiscent of the "rigidity" and topological character of the ground state of some strongly-correlated quantum systems that are currently investigated for topological quantum computation, namely quantum computation that is robust against dephasing and noise[24,25,26]. This analogy is not far-fetched. In fact, in the case of self-organising circuits, instantons, by connecting topologically-distinct critical points in the phase space, correlate elements of the circuit non-locally in space and time[27]. This non-locality is somewhat reminiscent of quantum entanglement. However, self-organising logic gates are circuits that **achieve long-range order without quantum-mechanical effects**.

[22] S. Coleman. Aspects of Symmetry, Chapter 7. Cambridge University Press, 1977.

[23] K. Hori, S. Katz, R. Klemm, A. Pandharipande, R. Thomas, C. Vafa, R. Vakil, and E. Zaslow. Mirror symmetry. Clay Mathematics, 2000.

[24] Michael Freedman, Alexei Kitaev, Michael Larsen, and Zhenghan Wang. Topological quantum computation. Bulletin of the American Mathematical Society, 40(1):31–38, 2003.

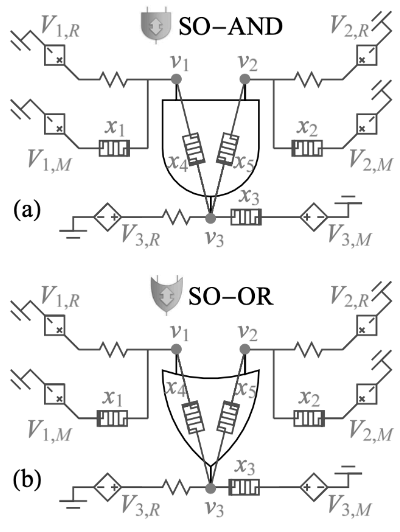
[25] Chetan Nayak, Steven H Simon, Ady Stern, Michael Freedman, and Sankar Das Sarma. Non-abelian anyons and topological quantum computation. Reviews of Modern Physics, 80(3):1083, 2008.

[26] A Yu Kitaev. Fault-tolerant quantum computation by anyons. Annals of Physics, 303(1):2–30, 2003.

[27] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. Ann. Phys. (Berlin), 529:1700123, 2017.

Self-organising logic gates: Dynex chip's building blocks

Let us start by outlining a model of self-organising logic gates as a system of **coupled, nonlinear, ordinary differential equation**s. We will then solve these equations numerically to identify instantons, their critical points, and, by diagonalising the Jacobian (the matrix of the derivative of the flow vector field), their topological features.



In this figure, we show the self-organising AND/OR gates we employ in our algorithm. They are modelled with standard resistors, resistors with memory (memristive elements)[28], and voltage- controlled voltage generators[29]. The memristive elements contain a capacitance in parallel, representing parasitic capacitive effects. The difference between the circuitry of the self-organising-AND and self-organising-OR gates is the orientation of the memristive elements.

We want these gates to **self-organise into the correct logical proposition** irrespective of the terminal to which the truth value is assigned. To better understand how this is accomplished, it is beneficial to start from a specific example. Let us then choose to encode the logical 1 (True) with 1 V and the logical 0 (False) with −1 V.

Consider first the self-organising-AND. If we set the voltage v1 to 1 V, the system should evolve to either v2 = v3 = 1 V or v2 = v3 = −1 V. Both are logically consistent with an AND truth table. On the other hand, if we consider the self-organising-OR gate, and fix v1 to −1 V (logical 0), the system should evolve to either v2 =v3 =−1V or v2 =v3 =1V. The final result will depend on the initial conditions, namely the initial values of all voltages and internal state variables.

[28] M. Di Ventra, Y.V. Pershin, and L.O. Chua. Circuit Elements With Memory: Memristors, Memcapacitors, and Meminductors. Proceedings of the IEEE, 97(10):1717–1724, Oct 2009.

[29] Fabio Lorenzo Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27:023107, 2017.

Below, we describe a set of dynamical equations that accomplishes the above tasks. For the evolution of the memristive state variables we choose an equation of motion of the form[30],

$$\frac{d}{dt}x_j = -\alpha h(x_j, v_{M_j}) g(x_j) v_{M_j},$$

where x_j is the state variable for the j-th memristive element. The function h serves to cutoff the dynamics of the state variable in certain regimes. We choose the conductance of these elements, $g(x) = ((R_{off} - R_{on})x + R_{on})-1$, where we set $R_{off} = 1\ \Omega$ and $R_{on} = 0.01\ \Omega$. Thus, $g(x)v_M$ is equal to the current flowing through a memristor. The voltage drop, v_M, is measured based on the orientation of the memristor: $v_M = v_a - v_b$, where v_b is measured from the thick-bar side of the electronic symbol for the memristor. The coefficient α is restricted to be positive, and we choose α = 60. The physical meaning of α is discussed in[31]. Finally, the values of the state variables are bounded, and are typically chosen to be $x \in [0, 1]$[32].

Ideally, in order to strictly enforce $x \in [0,1]$, $h(x,v_M)$ should be represented by step functions[31]. However, in practical realisations and numerical simulations, the step functions should be replaced by some differentiable function. We use[32],

$$h(x, v_M) = (1 - e^{-kx})\hat{\theta}^r\left(\frac{v_M}{2V_t}\right) + (1 - e^{-k(1-x)})\hat{\theta}^r\left(-\frac{v_M}{2V_t}\right),$$

where k = 2, and choose Vt = 0.1 V. The θˆ r function is defined as,

$$\hat{\theta}^r(y) = \begin{cases} 1 & y > 1 \\ \sum_{i=r+1}^{2r+1} a_i y^i & 0 \le y \le 1 \\ 0 & y < 0 \end{cases}$$

where we use the simplest case, r = 1. The coefficients can be found by requiring continuity and differentiability in y = 0 and y = 1. The coefficients for our implementation are $a_2 = 3$ and $a_3 = -2$. If we analyse the particular case discussed above, we fix, for both self-organising logic gates, the voltage generator on terminal 1, and we perform standard nodal analysis on terminals 2 and 3 to find

[30] Fabio Lorenzo Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27:023107, 2017.

[31] M. Di Ventra and Y. V. Pershin. On the physical properties of memristive, memcapacitive and meminductive systems. Nanotechnology, 24(25), 2013.

[32] Fabio Lorenzo Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27:023107, 2017.

$$C(-\frac{d}{dt}v_1 - 2\frac{d}{dt}v_2 + 2\frac{d}{dt}v_3) = -i_2 + (v_2 - v_3)g(x_5) + (-V_{2,M} + v_2)g(x_2) + \frac{-V_{2,R} + v_2}{R},$$

$$C(-3\frac{d}{dt}v_1 - 3\frac{d}{dt}v_2 + 4\frac{d}{dt}v_3) = -i_3 + (v_1 - v_3)g(x_4) + \frac{V_{3,R} - v_3}{R} + (v_2 - v_3)g(x_5) + (-v_3 + V_{3,M})g(x_3),$$

where the capacitance is C = 10^{-5} F and R = 1 Ω [2]. The voltage generators generate a voltage from the relation $V_{i,j} = b_1v_1 + b_2v_2 + b_3v_3 + dc_{gate}$, with dc_{gate} a constant voltage specific to each gate[32]. The coefficients, b_k, along with dc_{gate}, are given in[32]. Terminals 2 and 3 are floating, therefore, $i_2 = i_3 = 0$. Additionally, d/dt $v_1 = 0$, due to terminal 1 being attached to a voltage generator that is held constant. The role of the voltage generators is to inject a large current when the gate is in an inconsistent configuration, a small current otherwise.

By **solving numerically the equations** from above, with appropriate substitutions, we obtain precisely what we were after: **a consistent logical solution for the given gate**.

From computational problems to Dynex chips

Now, with the self-organising logic gates formulated with the concepts of above, we can build Dynex chips for solving a variety of use cases:

- Federated machine learning[33,34,35,36,37,38,39,40];

[33] Yang, Q., Liu, Y., Chen, T. and Tong, Y., 2019. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2), pp.1-19.

[34] Kaissis, G.A., Makowski, M.R., Rückert, D. and Braren, R.F., 2020. Secure, privacy-preserving and federated machine learning in medical imaging. Nature Machine Intelligence, 2(6), pp.305-311.

[35] Wahab, O.A., Mourad, A., Otrok, H. and Taleb, T., 2021. Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys & Tutorials*, *23*(2), pp.1342-1397.

[36] Konečný, J., McMahan, H.B., Ramage, D. and Richtárik, P., 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*.

[37] Li, T., Sahu, A.K., Talwalkar, A. and Smith, V., 2020. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, *37*(3), pp.50-60.

[38] Huang, L., Shea, A.L., Qian, H., Masurkar, A., Deng, H. and Liu, D., 2019. Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records. *Journal of biomedical informatics*, *99*, p.103291.

[39] Tan, K., Bremner, D., Le Kernec, J. and Imran, M., 2020, August. Federated machine learning in vehicular networks: A summary of recent applications. In *2020 international conference on UK-China emerging technologies (UCET)* (pp. 1-4). IEEE.

[40] Konečný, J., 2017. Stochastic, distributed and federated optimization for machine learning. *arXiv preprint arXiv:1707.01155*.

- Accelerating deep-learning[41,42,43], machine learning[44] and A.I.[45];
- Constraint satisfaction problems[46,47,48,49];
- Mixed integer linear programming[50,51,52,53];
- Quadratic unconstraint binary optimisation[54,55,56];
- Maximum satisfiability problem[57];
- Subset sum problems[58]; and
- Integer factorisation[59].

The applications are limitless and fulfil the requirement of an ever **growing demand** for **efficient computing power**. The self-organising logic gates can be used to construct **individual circuits** to compute these computational tasks efficiently. It has already been demonstrated that such systems perform **orders of magnitude faster** than traditional

[41] Manukian, H., Traversa, F.L. and Di Ventra, M., 2019. Accelerating deep learning with memcomputing. *Neural Networks*, *110*, pp.1-7.

[42] Shinde, P.P. and Shah, S., 2018, August. A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)* (pp. 1-6). IEEE.

[43] Balas, V.E., Roy, S.S., Sharma, D. and Samui, P. eds., 2019. *Handbook of deep learning applications* (Vol. 136). New York: Springer.

[44] Sammut, C. and Webb, G.I. eds., 2011. *Encyclopedia of machine learning*. Springer Science & Business Media.

[45] Mitchell, T.M. and Mitchell, T.M., 1997. *Machine learning* (Vol. 1, No. 9). New York: McGraw-hill.

[46] Brailsford, S.C., Potts, C.N. and Smith, B.M., 1999. Constraint satisfaction problems: Algorithms and applications. European journal of operational research, 119(3), pp.557-581.

[47] Arróyave, R., Gibbons, S.L., Galvan, E. and Malak, R.J., 2016. The inverse phase stability problem as a constraint satisfaction problem: Application to materials design. JOM, 68, pp.1385-1395.

[48] Naruse, M., Aono, M., Kim, S.J., Kawazoe, T., Nomura, W., Hori, H., Hara, M. and Ohtsu, M., 2012. Spatiotemporal dynamics in optical energy transfer on the nanoscale and its application to constraint satisfaction problems. Physical Review B, 86(12), p.125407.

[49] Roldan, E., Neágny, S., Le Lann, J.M. and Cortes, G., 2010. Constraint satisfaction problem for case-based reasoning adaptation: application in process design. In Computer Aided Chemical Engineering (Vol. 28, pp. 397-402). Elsevier.

[50] Richards, A. and How, J.P., 2002, May. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301) (Vol. 3, pp. 1936-1941). IEEE.

[51] Floudas, C.A. and Lin, X., 2005. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. Annals of Operations Research, 139, pp.131-162.

[52] Little, J.D., 1966. The synchronization of traffic signals by mixed-integer linear programming. Operations Research, 14(4), pp.568-594.

[53] Morais, H., Kádár, P., Faria, P., Vale, Z.A. and Khodr, H.M., 2010. Optimal scheduling of a renewable micro-grid in an isolated load area using mixed-integer linear programming. Renewable Energy, 35(1), pp.151-156.

[54] Lewis, M. and Glover, F., 2017. Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis. Networks, 70(2), pp.79-97.

[55] Neven, H., Rose, G. and Macready, W.G., 2008. Image recognition with an adiabatic quantum computer I. Mapping to quadratic unconstrained binary optimization. arXiv preprint arXiv:0804.4457.

[56] Alom, M.Z., Van Essen, B., Moody, A.T., Widemann, D.P. and Taha, T.M., 2017, May. Quadratic unconstrained binary optimization (QUBO) on neuromorphic computing system. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 3922-3929). IEEE.

[57] Creignou, N., 1995. A dichotomy theorem for maximum generalized satisfiability problems. Journal of Computer and System Sciences, 51(3), pp.511-522.

[58] LaMacchia, B.A., 1991. Basis reduction algorithms and subset sum problems.

[59] Brent, R.P., 2000. Recent progress and prospects for integer factorisation algorithms. In Computing and Combinatorics: 6th Annual International Conference, COCOON 2000 Sydney, Australia, July 26–28, 2000 Proceedings 6 (pp. 3-22). Springer Berlin Heidelberg.

algorithmic approaches on a wide variety of combinatorial optimisation problems[60,61,62,63,64].

As an **example**, we show a Dynex chip mapping a constraint satisfaction problem, specifically, a Boolean satisfiability problem[65]. The **formulation** a Dynex chip representing a SAT problem will be detailed, along with its importance in computational complexity theory. Other use cases follow the same concept and principles and can be derived given the equations of motion provided. Since Dynex chips are non-quantum systems, their **equations of motion** can be efficiently **integrated numericall**y.

The Boolean satisfiability problem[60] (SAT) is an important decision problem solved by determining if a solution exists to a Boolean formula. Apart from its academic interest, the solution of SAT instances is required in a **wide range of practical applications**, including, travel, logistics, software/hardware design, etc.[66,67]. The SAT problem has been studied for decades, and has an important role in the history of computational complexity theory. Computer scientists, while categorising the efficiency of algorithms, defined the NP class for difficult decision problems[68,69]. NP-completeness is not exclusive to SAT, with hundreds of other NP-complete problems ranging from those of academic interest (graph theory, algebra and number theory, mathematical programming) to industry application (network design, data storage and retrieval, program optimisation).

---

[60] F. L. Traversa, P. Cicotti, F. Sheldon, and M. Di Ventra. Evidence of exponential speed-up in the solution of hard optimization problems. Complexity, 2018:7982851, 2018.

[61] Massimiliano Di Ventra and Fabio L. Traversa. Perspective: Memcomputing: Leveraging memory and physics to compute efficiently. Journal of Applied Physics, 123(18):180901, 2018.

[62] Haik Manukian, Fabio L Traversa, and Massimiliano Di Ventra. Accelerating deep learning with memcomputing. Neural Networks, 110:1–7, 2019.

[63] F. L. Traversa and M. Di Ventra. Memcomputing integer linear programming. arXiv:1808.09999, 2018.

[64] Forrest Sheldon, Fabio L. Traversa, and Massimiliano Di Ventra. Taming a nonconvex landscape with dynamical long-range order: Memcomputing ising benchmarks. Phys. Rev. E, 100:053311, Nov 2019.

[65] Justyna Petke. Bridging Constraint Satisfaction and Boolean Satisfiability. Springer, 2015.

[66] Michael R. Garey and David S. Johnson. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

[67] Joao Marques-Silva. Practical applications of boolean satisfiability. In 2008 9th International Workshop on Discrete Event Systems, pages 74–80. IEEE, 2008.

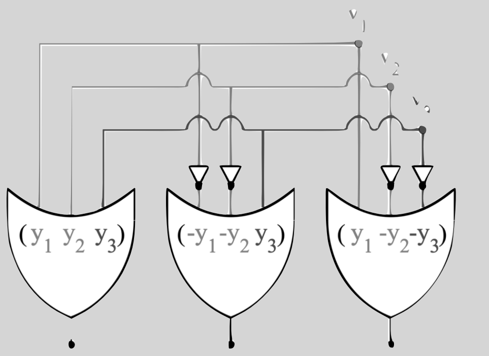
[68] Stephen A Cook. The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing, pages 151–158, 1971.

[69] Michael R. Garey and David S. Johnson. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

An instance of SAT is a Boolean formula with three components[70]:

1) A set of N Boolean **variables**: $y_1, y_2,...,y_N$.
2) A set of **literals**. A literal is a variable ($l = y$) or a negation of a variable ($l = \bar{y}$).
3) A set of M distinct **clauses**: $C_1, C_2,..., C_M$. Each clause consists of literals combined by logical OR connectives.

SAT is the decision problem of determining if an assignment of variables exists for which the Boolean formula returns TRUE, that is, all clauses must evaluate to TRUE as they are connected by AND operators to create the Boolean formula. If a such a solution exists, we say the SAT instance (Boolean formula) is satisfiable, otherwise, the instances is unsatisfiable. Commonly, it is said the instance is SAT or UNSAT, respectively.



Example of a Boolean circuit, in conjunctive normal form (CNF), representing a 3-SAT. The three OR clauses (seen inside the gates) are then converted to self-organising logic gates where the propositional variables y_i are represented as electrical voltages v_i. The traditional output of the self-organising-OR is forced to be true (logical 1), because all clauses must be true to satisfy a Boolean proposition in CNF. If a literal is the negation of a variable, then the associated "input" terminal on that gate must pass through a NOT gate (triangle symbol) before the terminal is connected to other terminals sharing the same variable.

The idea behind this approach is that the solutions of the SAT instance are mapped into the **equilibrium points** of a **dynamical system**. If the initial conditions of the dynamics belong to the **basin of attraction** of the equilibrium points, then the dynamical system will have to "fall" into these points. The approach is fundamentally different from the standard algorithms because dynamical systems perform computation in **continuous time**. Numerical simulation of continuous-time physical systems, an algorithm, requires the discretisation of time to integrate the ordinary differential equations (ODEs) representing the physical system. As such, the dynamical-systems approach is ideally suited for a hardware implementation, especially on **Graphic-Processing-Units** (GPU) with their capability of massively perform floating point operations in parallel.

[70] Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. Algorithms for the satisfiability (sat) problem. In Handbook of Combinatorial Optimization, pages 379–572. Springer, 1999.

The ability of continuous time dynamics to perform the solution search without resorting to chaotic dynamics results in **efficient simulations** (an algorithmic implementation) of Dynex chips using computationally-inexpensive integration schemes and modern computers. In addition, it was shown that such systems find the solution of a given problem by employing topological objects, known as **instantons**, that connect critical points of increasing stability in the phase space[71],[72]. Simulations found that they then **self-tune** into a critical (collective) state which persists for the whole transient dynamics until a solution is found[73]. It is this critical branching behaviour that allows them to explore **collective updates of variables** during the solution search, **without** the need to **check an exponentially-growing** number of states. This is in contrast to local-search algorithms which are characterised by a "small" (not collective) number of variable updates at each step of the computation[74].

To construct a Dynex chip that finds a satisfying assignment for SAT we follow the general procedure outlined in[75]. To begin, the Boolean variables, y_i, are transformed into continuous variables for use in the Dynex chip. The continuous variables can be realised in practice as voltages on the terminals of the self-organising OR gate. The gate can influence its terminals to push voltages towards a configuration satisfying its OR logic regardless of whether the signal received by the gate originates from the traditional input or the traditional output. The voltages are bounded, $v_i \in [-1, 1]$, with Boolean values recovered by thresholding: TRUE if $v_i > 0$, FALSE if $v_i < 0$, and ambiguous if $v_i = 0$. To perform the logical negation operation on the continuous variable, one trivially multiplies that quantity by −1. The self- organising logic circuit that comprises the Dynex chip is built by connecting all of the self-organising OR gates.

Next, we interpret a Boolean clause as a dynamical constraint function, with its state of satisfaction determined by the voltages. The m-th Boolean clause, $(l_{i,m} \vee l_{j,m} \vee l_{k,m})$, becomes a constraint function,

$$C_m(v_i, v_j, v_k) = \frac{1}{2}\min[(1 - q_{i,m}v_i), (1 - q_{j,m}v_j), (1 - q_{k,m}v_k)],$$

where $q_{i,m} = 1$ if $l_{i,m} = y_i$, and $q_{i,m} = -1$ if $l_{i,m} = \bar{y}_i$. The function is bounded, $C_m \in [0,1]$, and a clause is necessarily satisfied when $C_m < 1/2$. The instance is solved when $C_m < 1/2$ for all clauses. By thresholding the clause function we avoid the ambiguity associated with $v_i = 0$. If some voltage is ambiguous ($v_j = 0$) and all clauses are satisfied,

[71] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. Ann. Phys. (Berlin), 529:1700123, 2017.

[72] M. Di Ventra and Igor V. Ovchinnikov. Digital memcomputing: from logic to dynamics to topology. Annals of Physics, 409:167935, 2019.

[73] S. R. B. Bearden, F Sheldon, and M Di Ventra. Critical branching processes in digital memcomputing machines. EPL (Europhysics Letters), 127(3):30005, 2019.

[74] Alexander K. Hartmann and Heiko Rieger. New Optimization Algorithms in Physics. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2004.

[75] Massimiliano Di Ventra and Fabio L. Traversa. Perspective: Memcomputing: Leveraging memory and physics to compute efficiently. Journal of Applied Physics, 123(18):180901, 2018.

then any Boolean assignment to y_j will be valid in that configuration. The use of a minimum function in C_m preserves an important property of 3-SAT. A clause is a constraint, and, by itself, a clause can only constrain one variable (via its literal). The values of two literals are irrelevant to the state of the clause if the third literal results in a satisfied clause.

Finally, a Dynex chip employs memory variables to assist with the computation[74,75]. The memory variables transform equilibrium points that do not correspond to solutions of the Boolean formula into unstable points in the voltage space, leaving the solutions of the 3-SAT problem as the only minima. We chose to introduce two memory variables per clause: short-term memory, $x_{s,m}$, and long-term memory, $x_{l,m}$. The terminology intuitively describes the behaviour of their dynamics. For the short-term memory, $x_{s,m}$ lags C_m, acting as an indicator of the recent history of the clause. For the long-term memory, $x_{l,m}$ collects information so it can "remember" the most frustrated clauses, weighting their dynamics more than clauses that are "historically" easily satisfied. Both the number and type of memory variables, as well as the form of the resulting dynamical equations, are not unique provided neither chaotic dynamics nor periodic orbits are introduced[74].

We chose for the dynamics of voltages and memory variables the following,

$$\begin{aligned}
\dot{v}_n &= \sum_m x_{l,m} x_{s,m} G_{n,m} + (1+\zeta x_{l,m})(1-x_{s,m}) R_{n,m},\\
\dot{x}_{s,m} &= \beta(x_{s,m}+\varepsilon)(C_m-\gamma),\\
\dot{x}_{l,m} &= \alpha(C_m-\delta),
\end{aligned}$$

where the summation is taken over all constraints in which the voltage appears. The memory variables are bounded, with $x_{s,m} \in [0,1]$ and $x_{l,m} \in [1,10^{4M}]$. The boundedness of voltage and memory variables implies that there are no diverging terms in the above equations.

The parameters α and β are the rates of growth for the long-term and short-term memory variables, respectively. Each memory variable has a threshold parameter used for evaluating the state of C_m, and the two parameters are restricted to obey $\delta < \gamma < 1/2$. (This also guarantees that there is a sufficiently large basin of attraction for the solutions. The equations have a small, strictly-positive parameter, $0 < \varepsilon \ll 1$, to remove the spurious solution ($x_{s,m} = 0$). However, ε additionally serves as a trapping rate in the sense that smaller values of ε make it more difficult for the system to flip voltages when some C_m begins to grow larger than γ.

In the equations, the first term in the summation is a "gradient-like" term, the second term is a "rigidity" term[76]. The gradient-like term attempts to influence the voltage in a clause based on the value of the other two voltages in the associated clause,

[76] S. R. B. Bearden, F Sheldon, and M Di Ventra. Critical branching processes in digital memcomputing machines. EPL (Europhysics Letters), 127(3):30005, 2019.

$$G_{n,m}(v_n, v_j, v_k) = \frac{1}{2} q_{n,m} \min[(1 - q_{j,m} v_j), (1 - q_{k,m} v_k)].$$

Consider the two extremes: if the minimum results is $G_{i,m} = 1$, then v_i needs to be influenced to satisfy the clause. Conversely, if the minimum gives $G_{i,m} = 0$, then v_i does not need to influence the clause state. For the rigidity term, we choose

$$R_{n,m}(v_n, v_j, v_k) = \begin{cases} \frac{1}{2}(q_{n,m} - v_n), & C_m(v_n, v_j, v_k) = \frac{1}{2}(1 - q_{n,m} v_n), \\ 0, & \text{otherwise}, \end{cases}$$

The purpose of the three rigidity terms for a constraint is to attempt to hold one voltage at a value satisfying the associated *m*-th clause, while doing nothing to influence the evolution of the other two voltages in the constraint. Again, this aligns with the 3-SAT interpretation that a clause can only constrain one variable. The short-term memory variable acts as a switch between gradient-like dynamics and rigid dynamics. During the solution search, G_m will seek to influence three voltages until clause m has been satisfied. Then, as $x_{s,m}$ decays to zero, R_m takes over. The long-term memory variables weight the gradient-like dynamics, giving greater influence to clauses that have been more frustrated during the solution search. The rigidity is also weighted by $x_{l,m}$, but reduced by ζ.

It is important to realise that any simulation of a dynamical system is an algorithm because the continuous-time dynamics of the system must be discretised. The equations of motion of the individual Dynex chip constructed is numerically integrated with the **forward-Euler method** using an adaptive time step[77]. The number of possible initial conditions for a **parallel integration** on GPU is defined with the number of variables $2n$, representing the positive and negative occurrence of each variable, two polarities (positive and negative) as well as four stages:

Number of initial conditions (IC) = 16n

Comprehensive empirical studies of the integration method have demonstrated that the algorithm has an **upper bound complexity** of n^5 integration steps when all initial conditions are being simulated in parallel. Thus, to guarantee a solution of the computational problem, $16n$ Dynex chips have to be simulated in parallel for the maximum duration of n^5 integration steps.

[77] Lei, Z. and Hongzhou, J., 2012, December. Variable step euler method for real-time simulation. In *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology* (pp. 2006-2010). IEEE.

Collective behaviour (**long-range order**)[78,79] in the Dynex chips is responsible for the observed efficiency in the solution search. The dynamics described by the equations of motion terminate only when the system has found the solution to the 3-SAT problem (namely the phase space has **only saddle points** and the minima corresponding to the solution of the given problem. In addition, neither periodic orbits nor chaos can coexist if solutions of the 3-SAT are present. Finally, using supersymmetric topological field theory, it had been demonstrated that the **continuous-time dynamics** (physical implementation) reach the solution of a 3-SAT instance, for a fixed αr, in linear or sub-linear continuous time, irrespective of the difficulty of the instance.

DynexSolve Proof-of-Useful-Work

The DynexSolve mining algorithm **performs the numerical integration** of all Dynex chips required for the computational job and is therefore classified as a Proof-of-Useful-Work (PoUW) mining algorithm.

Depending on problem size (number of variables *n* and number of clauses *m*) and the memory available on the provided Graphic-Processing-Units (GPUs) the capacity for each miner to run **parallel Dynex chips** is determined. As all miners are working collectively on computational jobs, a job and chip scheduling system is required to assign and balance the work required:

The **Dynex Mallob system**, named after the term *malleable,* which defines a distributed computing environment[80,81], has been inspired by [82] and [83]. It dynamically assigns jobs with the respective available initial conditions to the individual miners and ensures that all 16*n* initial conditions are being computed for a maximum duration of n^5 integration steps.

DynexSolve combines two algorithms, namely the numerical integration of Dynex Chips as well as a modified CryptoNight hashing function to confirm blocks on the Dynex block chain. It has been designed to spend the **majority** of the computational **energy on the numerical integration** (meaningful work) to ensure that almost no resources are being wasted with hashing:

[78] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. Ann. Phys. (Berlin), 529:1700123, 2017.

[79] M. Di Ventra and Igor V. Ovchinnikov. Digital memcomputing: from logic to dynamics to topology. Annals of Physics, 409:167935, 2019.

[80] Desell, T., Maghraoui, K.E. and Varela, C.A., 2007. Malleable applications for scalable high performance computing. *Cluster Computing*, *10*, pp.323-337.

[81] Ghafoor, S.K., 2007. Modeling of an adaptive parallel system with malleable applications in a distributed computing environment. Mississippi State University.

[82] Schreiber, D. and Sanders, P., 2021. Scalable SAT solving in the cloud. In Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24 (pp. 518-534). Springer International Publishing.

[83] Sanders, P. and Schreiber, D., 2022. Mallob: Scalable SAT Solving On Demand With Decentralized Job Scheduling. *Journal of Open Source Software*, *7*(76), p.4591.

Algorithm: DynexSolve

```
1: Input: Computational problem := DynexMallob()
2: space := parallel Dynex chips fitting on GPU memory
3: available Dynex chips := DynexMallob(space)
4: Build Dynex Chip circuit for numerical integration = $n$
5: solved := false
6: for each CHIP do
7:     initial conditions$_{CHIP}$ = available Dynex chips
8:     while solved = false & integration steps$_{CHIP}$ < $n^5$ do
9:         integration steps$_{CHIP}$ := 0
10:        init state$_{hash}$, state$_{nonce}$ and state$_{diff}$
11:        pouw$_{blob}$ := nonce$_{\text{blockchain}}$ + timestamp + pouw$_{\text{state}}$
12:        while integration steps < batch-size do
13:            state$_{CHIP,n}$ < Adaptive Forward Euler step
14:            if solved = true then
15:                return (solved)
16:            end if
17:            state$_{hash}$, state$_{diff}$ := lighthash(pouw$_{blob}$, state$_{nonce}$)
18:            pouw$_{blob,loc}$ := localMinima(state)
19:            integration steps$_{CHIP}$ := +1
20:        end while
21:        DynexMallob(state)
22:        eligible$_{counter}$ := pouw$_{eligable}$
23:        hashing$_{counter}$ := 0
24:        while hashing$_{counter}$ < eligible$_{counter}$ do
25:            hash := CryptoNight$_{\text{modified}}$(blocktemplate, nonce)
26:            if hash$_{diff}$ > block$_{diff}$ then
27:                submit pouw$_{blob}$ + pouw$_{hash}$ + nonce
28:            end if
29:            hashing$_{counter}$ := +1
30:        end while
31:    end while
32: end for
```

Initially, DynexSolve **retrieves the definition** of the assigned **computational task** from the Dynex Mallob system, which allows to calculate the total capacity of all parallel Dynex Chips fitting on the connected Graphic-Processing-Units (GPUs), defined as *space*.

```
1: **Input**: $Computational\ problem := \text{DynexMallob()}$
2: $space :=$ parallel Dynex chips fitting on GPU memory
```

Given *space*, DynexSolve retrieves the set of assigned Dynex Chips from the Dynex Mallob system which also define the initial conditions for the Dynex chips to compute. Based on that data, DynexSolve **builds the corresponding system** of equations of motions to be numerically integrated.

```
3: $available\ Dynex\ chips := \text{DynexMallob}(space)$
4: Build $Dynex\ Chip\ circuit$ for numerical integration $= n$
```

Every chip is integrated **in parallel** on each of the connected Graphic-Processing-Units with the different initial conditions provided, as long as either a solution was found or the maximum number of integration steps n^5 has been reached.

```
5: $solved := false$
6: **for each** $CHIP$ **do**
7:         $initial\ conditions_{CHIP} = available\ Dynex\ chips$
8:         **while** $solved = false\ \&\ integration\ steps_{CHIP} < n^5$ **do**
```

The numerical integration is performed in **batches** (typically 10,000 integration steps per batch per Dynex chip). Every integration step creates a **unique state** for each Dynex chip:

```
13:                 $state_{CHIP,n} < Adaptive\ Forward\ Euler\ step$
```

The algorithm also performs hashing on the provided $PoUW_{blob}$ as well as the calculation of the current energy landscape (local minima):

```
17:                 $state_{hash}, state_{diff} := \text{lighthash}(pouw_{blob}, state_{nonce})$
18:                 $pouw_{blob,loc} := localMinima(state)$
```

This ensures that the performed work is **unique** and **accurate**. The current energy landscape given a current state can be verified quickly with a simple function call for any given computational job.

Each initial condition has a different number of integration steps required to reach the lowest possible global energy level of the underlying computational problem, which represents a ***solution***. As soon as a solution to the job has been found, the Dynex Mallob system is being updated, the solution state submitted to Dynex and the job marked as "finalised".

```
14:         if $solved = true$ then
15:             return $(solved)$
16:         end if
```

The successful PoUW work continuously defines the overall hash-rate of the miner, also determining the number of eligible hashes DynexSolve can use for the blockchain related $\text{CryptoNight}_{\text{modified}}$ hashing function to confirm blocks in the Dynex block chain:

```
22:     $eligible_{counter} := pouw_{eligable}$
23:     $hashing_{counter} := 0$
24:     while $hashing_{counter} < eligible_{counter}$ do
25:         $hash := \text{CryptoNight}_{\text{modified}}(blocktemplate, nonce)$
26:         if $hash_{diff} > block_{diff}$ then
27:             submit $pouw_{blob} + pouw_{hash} + nonce$
28:         end if
29:         $hashing_{counter} := +1$
30:     end while
```

Submitted blocks to the Dynex block chain require a **successful verification** from its PoUW data as well as the calculated block nonce itself. Per definition of the algorithm, both are **entangled** and **uniquely connected** to the underlying **computational job,** which guarantees that block nonces can be found only if the entire Proof-of-Useful-Work scheme has been performed with the DynexSolve algorithm.

Mining renumeration scheme

The renumeration scheme for DynexSolve mining consists of the following elements:

(i) ***Block reward***: Miners who are finding a block nonce of a block are receiving the block reward for this block. The block reward follows a smooth emission curve[84].

(ii) ***Transaction fees***: In addition to the block reward, also the transaction fees included in the mined block are being rewarded.

(iii) ***Block fees***: Dynex customers who post and run computation jobs on the Dynex platform can define a block fee they are paying for the computations. The *block fee* is paid for every block which is being mined working on the computational problem. As a) the network hash-rate, b) the maximum complexity ($16n$) and c) the upper bound of required integration steps (n^5) are known at job creation, customers can allocate a pre-defined amount for any job. The *Dynex Mallob system* is assigning highest paid jobs first, then in descending order. The *block fee* is rewarded to the miners similarly as the block reward. This guarantees **continuity and sustainability** of the **business model** for miners, also for the period when all blocks have been mined.

(iv) ***Solution reward***: As an additional motivation for miners, Dynex customers can define a *solution reward* for the miner who completed the computational job first. It is being automatically rewarded to the first miner completing a job. This incentivises miners to continuously perform DynexSolve PoUW calculations (rather than often restarting jobs) and also provides CPU miners a valid chance to win the solution reward. In contrast to GPU miners, where the initial conditions are pre-defined in the unique Dynex chips, are the initial conditions for CPU miners randomised.

[84] https://dynexcoin.org/wp-content/uploads/2023/01/Dynex-whitepaper.pdf

Additional Ressources

Main Dynex website:
https://dynexcoin.org/

Dynex white paper:
https://dynexcoin.org/wp-content/uploads/2023/01/Dynex-whitepaper.pdf

Dynex in 10 Layman's Terms:
https://dynexcoin.org/dynex-in-ten-laymans-terms/

DynexSolve: Proof-of-Useful-Work (PoUW):
https://dynexcoin.org/discover-dynex/

Dynex History & Roadmap:
https://dynexcoin.org/discover-dynex/#roadmap

Dynex Mining Pools on MiningPoolStats:
https://miningpoolstats.stream/dynexcoin

Dynex Mining Pools Certification Status:
https://dynexcoin.org/mining-pool-certifications-status/

Dynex Mining Software (DynexSolve):
https://github.com/dynexcoin/Dynex/releases/tag/DynexSolve

Dynex Node and CLI Wallet:
https://github.com/dynexcoin/Dynex/releases/tag/Dynex_2.2.2

Dynex Blockchain Explorer:
https://dynex.dyndns.org/home.php

Dynex Bounty Program:
https://dynexcoin.org/dynex-bounties/

Dynex Wallets:
https://dynexcoin.org/get-dnx/#wallets

Dynex Mobile Web Wallet:
https://wallet.dynexcoin.org/

Dynex Introductory Video:
https://dynexcoin.org/video/